基于 HBase 的细粒度访问控制方法研究 *

黄良强，朱 焱，陶 霄
(西南交通大学 信息科学与技术学院，成都 611756)

摘 要：为增强 HBase 的安全访问控制能力，提出一种针对 HBase 的细粒度访问控制方法。该方法通过修改优化 HBase 源码，扩展访问控制权限、重写访问控制器达到细粒度访问控制的目的。归纳出应用于 HBase 的 RBAC 模型，内建数据库角色以解决权限扩展后细粒度权限管理难度增大的问题。通过设计实验测试用例，验证了提出的细粒度访问控制方法能更全面地保护 HBase 数据，解决了原有方法带来的权限过粗的问题，降低了数据可能被恶意地执行修改、删除等操作所带来的巨大安全风险。

关键词：HBase 数据库；访问控制；细粒度权限；数据库角色
中图分类号：TP309.2 doi: 10.19734/j.issn.1001-3695.2018.08.0648

Research on fine-grained access control method based on HBase

Huang Liangqiang, Zhu Yan, Tao Xiao
(*School of Information Science & Technology, Southwest Jiaotong University, Chengdu* 611756, *China*)

Abstract: In order to enhance the access control ability of HBase, this paper proposed a fine-grained access control method for HBase. This method achieves the purpose of fine-grained access control by modifying and optimizing the HBase source code, extending the access control permissions, and rewriting the AccessController. Moreover, this paper generalizes the RBAC model that applied in HBase, and use built-in database roles to solve the problem that fine-grained permissions management becomes more difficult after extending permissions. By designing experimental test cases, it is verified that the proposed fine-grained access control method can protect HBase data more comprehensively. This paper solves the problem that excessive permissions caused by the original method, and reduces the huge security risk caused by data may be maliciously performing operations such as modification and deletion, etc.

Key words: HBase; access control; fine-grained permissions; database role

0 引言

NoSQL 数据库因其对海量数据有着高效的管理能力，广泛应用于各类大数据生产场景，例如，阿里巴巴集团早在 2011 年就开始把 HBase 投入到上千台节点的集群中用于淘宝历史交易记录、蚂蚁安全风控等大数据的存储。然而，其安全性问题也被业界广泛关注。大多数 NoSQL 数据库只是提供简单的数据保护支持，无法满足实际生产中对数据安全性能的需求，特别是对较敏感数据的保护。例如，在医院大数据应用场景中，病人的姓名、疾病史等敏感信息是需要被重点保护的数据对象。HBase 作为一种 NoSQL 列式存储数据库技术，同样存在以上安全性问题，因此，研究如何增强 HBase 的访问控制能力成为一种迫切需求。

关系型数据库在访问控制技术理论与应用方面更加成熟完备。占据着数据库引擎排行榜[1]前三的关系型数据库 Oracle、MySQL、Microsoft SQL Server 都采用基于角色的访问控制（RBAC）技术，实现了表级、列级等细粒度对象，插入、修改、删除等细粒度操作两方面的访问控制。而新诞生的 NoSQL 数据库类型多样，例如，列式存储的 HBase、Cassandra，文档存储的 MongoDB、Couchbase，键值对存储的 Redis、Memcached，图存储的 Neo4j、Microsoft Azure Cosmos DB。不同类型的 NoSQL 数据库采用不同的架构模型，实现访问控制的方法也不尽相同。

MongoDB 采用 RBAC 技术，分为内建数据库角色和用户自定义角色两种。内建角色提供了数据库系统中常用的权限集合，权限粒度较粗，一个权限包括一系列操作；自定义角色可实现细粒度权限授权，但是权限操作范围只是控制在数据库或集合级别，没有进一步深入对集合中字段的访问控制。Microsoft Azure Cosmos DB 使用主密钥和资源令牌进行身份验证，资源令牌与数据库中的权限相关联，确定用户是否对数据库中的应用程序资源拥有访问权限（读写、只读或无访问权限），应用程序资源包括集合、文档、附件、存储过程、触发器和用户定义函数，身份验证时，使用资源令牌允许或拒绝访问资源[2,3]。

HBase 提供六种简单的权限[4]，分别是 Superuser、Admin(A)、Create(C)、Write(W)、Read(R)、Execute(X)。Superuser 作为超级用户，可执行任意数据库操作;因此可授予普通用户的只能是另外五种权限。通过分析 HBase ACL Matrix 和实验验证发现，如果某个用户拥有 Write 权限，则该用户可以执行如 *put*、*delete*、*append*、*increment* 等写操作。在实际生产应用中，若要控制用户只能写入数据而不能删除数据，按照 HBase 现有的粗粒度权限授权则无法满足这样的

收稿日期：2018-08-06；**修回日期：**2018-10-12 **基金项目：**四川省学术和技术带头人后备人选科研基金资助项目（WZ0100112371408, YH1500411031402）；四川省学术和技术带头人科研基金资助项目（WZ0100112371601/004）；四川省科技服务业示范项目（2016GFW0166）

作者简介：黄良强（1992-），男，重庆大足人，硕士研究生，主要研究方向为大数据访问控制技术（hwanglq@163.com）；朱焱（1965-），女，广西桂林人，教授，博士，主要研究方向为数据挖掘、Web 异常模式发现、大数据管理与智能分析；陶霄（1995-），男，山东威海人，硕士研究生，主要研究方向为数据库与数据挖掘.

操作控制需求。针对以上 HBase 的问题，本文的研究目的和意义在于解决 HBase 原有粗粒度权限控制方法的弊端，实现细粒度权限控制，增强 HBase 的安全访问控制能力。

关于 NoSQL 数据库细粒度访问控制方法的研究，Colombo 等人[5]提出了一种通用的针对 NoSQL 数据库的基于属性的访问控制（ABAC）方法，该方法借助 SQL++技术[6]实现，在 Couchbase 4.5 平台上验证了所提出方法的可行性。然而，SQL++仅支持如 MongoDB、Couchbase、JSONiq、Hive 等 NoSQL 或 NewSQL 数据库，不支持 HBase；此外，他们也未说明在不支持 SQL++的 NoSQL 数据库平台上如何实施提出的细粒度访问控制方法。文献[7]存在许多与文献[5]相同的问题，尽管作者声称提出的方法适用于 NoSQL 数据库，却未解释如何在 NoSQL 数据库平台上实现。

文献[8]在 MongoDB 中集成使用了 RBAC、ABAC 和基于内容的访问控制技术，把 MongoDB 定制成为一个私有的数据库。文献[9]中，蔡平在研究 HBase 的安全性时，提出通过修改 HBase 源码来拓展访问控制权限的方法是可行的，但是作者仅仅任意设定了三个权限加以验证提出的方法的可行性，没有给出如何划分细粒度权限的依据，也没有探讨可能出现的问题并给出解决方案。文献[10]中，Lai 等人在保留 HBase 现有访问控制权限的条件下，按照 HBase Shell 命令的读、写等类别为依据划分细粒度权限，采用 XACML 技术来表达用户角色授权信息，通过定制观察者类型的协处理器[11]来实现提出的细粒度访问控制方法。文献[10]的不足之处在于划分细粒度权限的依据不够严谨，没有考虑划分的细粒度权限之间是否存在权限依赖关系等问题。

本文针对 HBase 提出的细粒度访问控制方法更全面完整，主要包括：a）分析了 HBase 现有的访问控制方法，理清 Shell 命令类别、Shell 命令与 Java API 的调用关系，给出细粒度权限划分依据;b）通过归纳应用于 HBase 的 RBAC 模型和内建数据库角色，解决了细粒度权限划分造成的权限管理难度增大的问题;c）通过修改优化源代码，扩展权限列表和重写访问控制器，实现所提出的细粒度访问控制方法;d）设计测试用例，验证所提出的细粒度访问控制方法的有效性。

1　细粒度访问控制方法设计

1.1　HBase 访问控制框架

HBase 访问控制框架主要包括如下三个模块：

a)操作接口模块,它是 HBase 提供给用户执行数据库操作和授权/撤权操作的接口，有 Shell 命令和 Java API 两种客户端方式。

b)访问控制模块，用于拦截用户的操作请求。执行授权/撤权操作时，该模块与权限存储模块交互，写入/删除授权信息；执行数据库操作时，该模块用于判定用户是否拥有执行操作请求的权限。

c)权限存储模块，用于存储用户和角色的授权信息。HBase 采用访问控制列表（ACL）的方式实现，ACL 的表模式如表 1 所示。HBase 缺省使用名称为 default 的命名空间（Namespace，等同于关系型数据库管理系统中的数据库），使用@字符区分是命名空间级别授权还是命名空间下的低级别授权，也使用@字符区分是对用户授权还是对角色授权。

表 1　ACL 的表模式

Table 1　Table schema of ACL

行键	数据
<@命名空间|命名空间:表|表>	列族:{<用户|@角色> [,列族[,列]]: '权限集合', ……}

1.2　权限控制级别与权限判定算法

HBase 提供五种权限控制级别，如图 1 所示。五种级别呈现包含关系，优先级从全局到列依次降低，低级别自动继承上级的授权。例如：授予用户 *A* 在命名空间 *NS* 上有“读”权限，则 *A* 在该 *NS* 下的所有树节点上都有“读”权限。

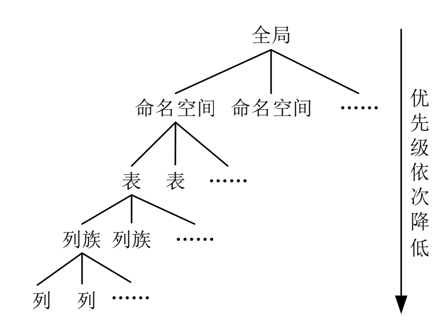


图 1　权限控制级别与优先级

Fig. 1　Permission control level and priority

用户提交操作请求后，HBase 向集群节点分发操作请求并执行。在操作请求执行前，访问控制器会捕获操作请求，并判定该用户是否具有执行操作请求的权限。

权限判定算法描述如下：

参数定义：*ul*=用户∪角色组；*ns*=命名空间名称；*table*=表名称；*cf*=列族名称；*cq*=列名称；*action*=执行操作请求所需的权限。

输入参数列表：<ul [, ns [, table [, cf [, cq]]]], action>

输出：*true* 表示权限判定通过；抛出权限异常信息表示判定失败，拒绝执行操作请求。

```
for i=0 to ul.length do
    全局级权限匹配，参数列表<ul[i], action>
    if 全局级匹配成功 then
        结束循环，返回 true
    end if
    if ns 不为空 then
        命名空间级权限匹配，参数列表<ul[i], ns, action>
        if 命名空间级匹配成功 then
            结束循环，返回 true
        end if
    end if
    if table 不为空 then
        表级权限匹配，参数列表<ul[i], ns, table, action>
        if 表级匹配成功 then
            结束循环，返回 true
        end if
    end if
    if cf 不为空 then
        列族级权限匹配，参数列表<ul[i], ns, table, cf, action>
        if 列族级匹配成功 then
            结束循环，返回 true
        end if
    end if
    if cq 不为空 then
        列级别权限匹配，参数列表<ul[i], ns, table, cf, cq, action>
        if 列级匹配成功 then
            结束循环，返回 true
        end if
    end if
end for
```

抛出权限异常信息

算法主体中，权限匹配规则是根据输入参数在 ACL 表中查找是否存在相同的授权信息，有则匹配成功，否则匹配失败。

1.3 细粒度权限设计

API 是 HBase 提供给用户原子性的操作方式，Shell 命令是基于 API 编写的功能封装。例如，Shell 命令 *truncate* 的功能是清空表内数据，保留表的属性。为完成该任务，Shell 命令的执行步骤如下：

a)执行获取表描述信息操作。

b)执行判断表是否停用操作。

c)如果表已停用则执行清空表内数据操作；否则终止任务执行。

d)执行删除表操作。

e)按照步骤 a）获取得到的表描述信息执行重建表操作。其中，每一步操作调用一个 API，如步骤 a）调用 *getTableDescriptor*，该 API 对应的 Shell 命令为 *describe*。命令 *truncate* 调用了 5 个 API，可见 Shell 命令会调用一个或多个 API 来完成对应的任务。此外，Shell 命令没有覆盖所有的 API，例如用于关闭集群的 *stopMaster*、*shutdown* 等 API。如果按照文献[10]中对照 shell 命令划分细粒度权限，即一个 shell 命令划分为一种权限的方法，所划分出的细粒度权限可能无法控制相关 API，细粒度权限划分不完整以及部分细粒度权限存在依赖关系，例如，权限'truncate'会依赖'describe'、'drop'和'create'权限，从而造成 HBase 安全隐患。

针对细粒度权限划分不完整、部分细粒度权限之间存在依赖关系的问题，本文提出一种按 API 进行细粒度权限划分的方法，从 API 的角度实现细粒度访问控制，保证权限划分完整、权限之间不存在依赖关系。

首先参照 SQL 语句中数据定义语句（DDL）、数据操纵语句（DML）等命令集合，把会造成 HBase 安全隐患的 API 按照其功能类型进行归类，共 8 种 API 类别，如表 2 所示。其目的在于能根据 API 类别更加集中地管理归类前凌乱的 API。

表 2 HBase 中 API 类别

Table 2 API types in HBase

类别	API 集合
DML	*appendAfterRowLock, append, delete, getOp, increment, checkAndDelete, put, checkAndPut, ……*
DDL	*modifyTable, addColumn, modifyColumn, deleteColumn, createTable, disableTable, deleteTable, enableTable, ……*
Security	*grant, revoke, getUserPermissions*
Snapshot	*snapshot, listSnapshot, cloneSnapshot, restoreSnapshot, deleteSnapshot*
Namespace	*createNamespace, deleteNamespace, modifyNamespace, getNamespaceDescriptor, listNamespaceDescriptors*
Tools	*move, assign, unassign, regionOffline, balance, balanceSwitch, flushTable, compact, openRegion, closeRegion, split, shutdown, stopMaster, mergeRegions, stopRegionServer, ……*
Quotas	*setUserQuota (global level), setUserQuota(namespace level) , setNamespaceQuota, setTableQuota, ……*
Endpoint	*invoke*

为保证划分的细粒度权限在实际应用中更具合理性，对表 2 进一步分析，可把各类别中相近功能的 API 归结为一种具体操作，例如，DDL 类别中的 *modifyTable*、*addColumn*、*modifyColumn*、*deleteColumn* 可归结为 ALTER 操作。本文在表 2 的基础上共归结出 8 种类别，46 种具体操作，如表 3 所示。

表 3 API 类别与具体操作集合

Table 3 API types and concrete operation sets

类别	具体操作集合
DML	APPEND, DELETE, GET, INCR, TRUNCATE, SCAN, PUT
DDL	ALTER, CREATE, DESCRIBE, DISABLE, ENABLE, LIST, DROP
Security	GRANT, REVOKE, USER_PERMISSION
Snapshot	LIST_SNAPSHOT, CLONE_SNAPSHOT, RESTORE_SNAPSHOT, DELETE_SNAPSHOT, SNAPSHOT
Namespace	CREATE_NAMESPACE, DROP_NAMESPACE, ALTER_NAMESPACE, DESCRIBE_NAMESPACE, LIST_NAMESPACE
Tools	MOVE, ASSIGN, UNASSIGN, OFFLINE, SPLIT, BALANCER, COMPACT, SHUTDOWN, BALANCE_SWITCH, STOP_MASTER, FLUSH, MERGE_REGION, WAL_ROLL, BULKLOAD, OPEN_REGION, CLOSE_REGION, STOP_REGION_SERVER
Quotas	QUOTA
Endpoint	EXECUTE

所提出的细粒度权限划分依据是：将造成 HBase 安全隐患的 API 归为多个类别，每个类别划分为多个具体操作，一种具体操作划分为一种权限。根据这种划分思想，共划分为 8 种权限类别，46 种细粒度权限，部分权限类别与权限集合如表 4 所示。

表 4 中，授权级别单元格值 G 表示全局级、N 表示命名空间级、T 表示表级、CF 表示列族级、CQ 表示列级。

表 4 部分权限类别与权限集合

Table 4 Partial permission types and permission sets

(a) DML 权限类别

(a) DML permission type

权限名称	权限控制的功能	授权级别
APPEND	单元格拼接新值	G|N|T|CF|CQ
DELETE	删除单元格数据	G|N|T|CF|CQ
GET	获取某行记录	G|N|T|CF|CQ
INCR	计数器单元格值做加减	G|N|T|CF|CQ
PUT	新增记录	G|N|T|CF|CQ
SCAN	扫描表数据	G|N|T|CF|CQ
TRUNCATE	清空表数据	G|N|T

(b) DDL 权限类别

(b) DDL permission type

权限名称	权限控制的功能	授权级别
ALTER	更改表/列族定义	G|N|T|CF
CREATE	创建表	G|N
DESCRIBE	获取表描述信息	G|N|T
DISABLE	停用表	G|N|T
DROP	删除表	G|N|T
ENABLE	启用表	G|N|T
LIST	列出所有表	G|N|T

(c) Security 权限类别

(c) Security permission type

权限名称	权限控制的功能	授权级别
GRANT	授予用户权限	G|N|T|CF|CQ
REVOKE	撤销用户权限	G|N|T|CF|CQ
USER_PERMISSION	列出用户权限	G|N|T|CF|CQ

1.4 调整授权命令语法

HBase 提供 grant 方法授予用户权限，使用 revoke 方法撤销用户权限。授权命令语法格式如图 2 所示，而撤权命令语法格式为图 2 中 grant 替换为 revoke，并去掉权限集合项。

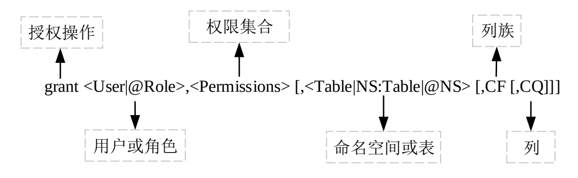


图 2 HBase 授权命令语法格式

Fig. 2 Authorization command syntax format of HBase

命令语法中使用@字符表示对角色授权，也使用@字符表示在命名空间级别授权。HBase 原有的五种权限，分别用字符'A' 'C' 'R' 'W' 'X'表示，图 2 中的权限集合项是五种权限的组合，例如：'CR'表示授予用户 CREATE 和 READ 权限。

本文扩展设计的细粒度权限个数较多，字符方式表示会使得权限组合更复杂，意义不明确。为此，本文采用权限全称，逗号隔开的方式，例如：'DELETE,CREATE,DROP'表示授予用户 DELETE、CREATE 和 DROP 权限。

2 应用于 HBase 的 RBAC 模型

本文 1.3 节扩展的细粒度权限带来了权限控制更具体，覆盖面更广的优势，也造成了当用户数量增大，权限管理难度增大的问题，本文深入研究了 HBase 引入的 RBAC 模型，并通过内建数据库角色集中管理扩展后的细粒度权限。

2.1 HBase 用户组机制

HBase 用户组机制依赖于 Hadoop 的用户组机制，Hadoop 的用户组机制依赖于 Linux/UNIX 操作系统的用户组机制。把用户组视为角色，则构建起 HBase 中用户与角色的所属关系。因此，基于 HBase 用户组机制实现 RBAC 模型，需在 Linux/Unix 操作系统级别设置用户和用户组信息。

HBase 获取到用户信息之后，根据用户名在操作系统中获取到所属的用户组集合作为该用户的角色组。

2.2 HBase 中的 RBAC 模型

参照 RBAC96 模型[12]，结合 2.1 节对 HBase 用户组机制的分析，本文归纳出如图 3 所示的应用于 HBase 的 RBAC 模型。

关于图 3 的 RBAC 模型的定义如下：

四元组(用户，角色，权限，会话)，即 U,R,P,S；

权限分配 $PA \subseteq P \times R$，角色与权限是多对多的关系；

用户分配 $UA \subseteq U \times R$，用户与角色是多对多的关系；

$user: S \rightarrow U$,每一个会话 s_i 映射到一个单用户 $user(s_i)$；$roles: S \rightarrow 2^R$，每一个会话 s_i 映射到一个角色子集 $roles(s_i) \subseteq \{r | (user(s_i), r) \in UA\}$，并且会话 s_i 具有权限 $\bigcup_{r \in roles(s_i)} \{p | (p,r) \in PA\}$；

通过角色可把用户与权限联系起来，会话是一个用户对多个角色的映射，即一个用户激活某个角色子集，用户权限是激活的各角色权限的并集[12,13]。

2.3 内建数据库角色

关系型数据库 Microsoft SQL Server 是最成熟的数据库产品之一，它在访问控制模块中能做到细粒度权限的访问控制，采用了 RBAC 模型，通过内建数据库角色集中管理细粒度权限。MongoDB 通过内建数据库角色来管理常用的权限集合。本文参考 SQL Server 和 MongoDB 在实际生产中积累的数据库权限管理经验，内建如表 5 所示的数据库角色，共五种角色类型。

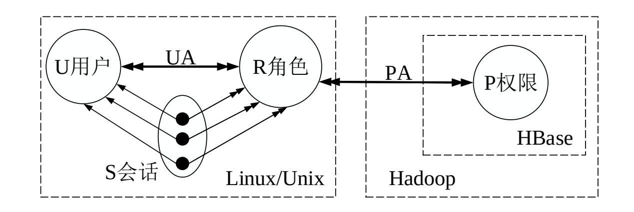


图 3 应用于 HBase 的 RBAC 模型

Fig. 3 RBAC model that applied in HBase

表 5 内建数据库角色

Table 5 Built-in database roles

角色	描述	权限
db_owner	拥有数据库中所有权限	命名空间级别所有权限
db_securityadmin	授予、回收角色/用户权限	GRANT, REVOKE, USER_PERMISSION
db_ddladmin	在数据库中运行任何 DDL 指令	DDL 类别所有权限
db_datawriter	添加、删除或更改数据	PUT, DELETE, APPEND, INCR
db_datareader	读取所有数据	GET, SCAN, GET_COUNTER

3 功能实现与测试

3.1 实验环境

实验在 3 台 PC 机搭建的集群上进行，环境配置如下：Ubuntu 16.04 LTS，HBase1.2.6，Hadoop 2.7.6，Zookeeper 3.10.4，JDK 1.8。集群为主从节点架构，包含 1 个主节点，2 个从节点。

3.2 改写源码集成细粒度访问控制功能

HBase 授权/撤权功能是向 ACL 表写入/删除授权信息。如图 4 所示是授权操作执行流程。源码修改优化需完成以下内容：

a）在权限列表中，增加 2.3 节划分的细粒度权限。HBase 源码中使用 Permission 类定义枚举类型变量 Action 来保存权限列表，使用 Byte 数据类型保存权限值。

b）重写授权请求封装接口，保证扩展的细粒度权限能成功写入到授权请求中。为缓解客户端的压力，HBase 把访问控制器部署在集群的每个节点上。客户端接收到授权命令后按照授权内容封装请求，然后通过 HBase 的二层查询技术找到操作请求执行的节点，将操作请求发送到该节点后，访问控制器将捕获操作请求，判定是否有 GRANT 权限，进而控制是否执行授权操作。

c）重写访问控制器。访问控制器继承自协处理器，提供如 prePut、preDelete 等预处理方法拦截数据库操作 put、delete 等，因此可在 prePut、preDelete 等方法中执行权限判定算法，达到细粒度权限访问控制的目的。

3.3 测试方法与测试结果

本文的测试工作包括三项目标:a)测试 2.3 节设计的所有细粒度权限；b)测试所设计的细粒度权限在各授权级别上是

否能真实有效地达到访问控制的目的；c)测试低优先级能否继承上级的授权。

本文编写客户端程序调用 HBase 的 API 进行测试。

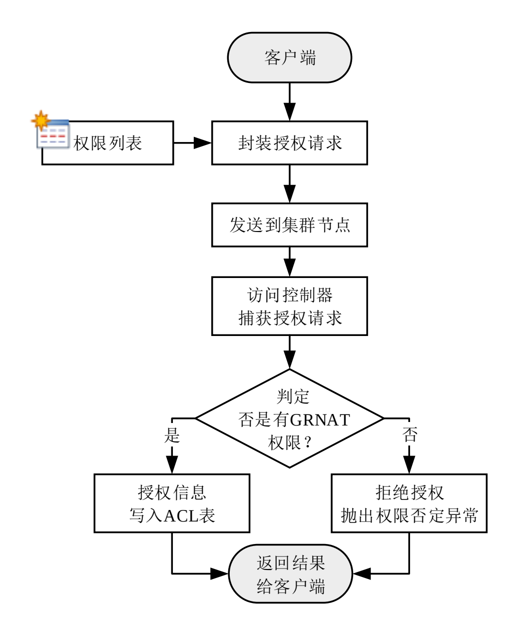


图 4　授权操作执行流程

Fig. 4　Authorized operation execution process

3.3.1 测试 Ⅰ

以 PUT 权限为例，PUT 权限用于控制数据库新增记录操作， API 名称为 *put* 和 *checkAndPut*。操作定义：*op1* 表示用户 *u1* 调用 *put* API 向命名空间 *ns1* 中表 *t1* 的列族 *cf1* 的列 *color* 新增一条记录。

测试方法描述如下：

a)不授予 *u1* 任何权限，*u1* 不拥有任何角色。执行 *op1* 操作，系统应抛出如图 5 所示的异常信息，否则访问控制失败。

b)授予 *u1* 在全局级别上有 PUT 权限，执行 *op1* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

c)撤销 *u1* 在全局级别的 PUT 权限，授予 *u1* 在命名空间 *ns1* 级别有 PUT 权限。执行 *op1* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

d)撤销 *u1* 在命名空间 *ns1* 的 PUT 权限，授予 *u1* 在命名空间 *ns1* 中表 *t1* 级别有 PUT 权限。执行 *op1* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

e)撤销 *u1* 在命名空间 *ns1* 中表 *t1* 级别的 PUT 权限，授予 *u1* 在命名空间 *ns1* 中表 *t1* 的列族 *cf1* 级别有 PUT 权限。执行 *op1* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

f)撤销 *u1* 在命名空间 *ns1* 中表 *t1* 的列族 *cf1* 级别的 PUT 权限，授予 *u1* 在命名空间 *ns1* 中表 *t1* 的列族 *cf1* 的列 *color* 级别有 PUT 权限。执行 *op1* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

g)撤销 *u1* 所有权限，分配 *u1* 拥有角色 *role1*。用户授权替换为角色授权，重复 a)~f)测试过程。

```
org.apache.hadoop.hbase.security.AccessDeniedException:
Insufficient permissions (user=u1, scope=ns1:t1, family=cf1:color,
params=[table=ns1:t1,family=cf1:color], action=PUT)
```

图 5　权限判定失败返回异常信息

Fig. 5　Permission judge failed to return exception information

如表 6 所示为执行 *op1* 操作时，PUT 权限判定通过与否，*P* 表示通过，*R* 表示拒绝。

表 6　PUT 权限控制结果

Table 6　PUT permission control result

	全局	命名空间	表	列族	列
无 PUT 权限	*R*	*R*	*R*	*R*	*R*
有 PUT 权限	*P*	*P*	*P*	*P*	*P*

同理，对于 DDL 权限类别中的 CREATE 权限，其功能是控制创建表操作。表 7 所示为用户 *u1* 在命名空间 *ns1* 中执行创建表 *t2* 操作时，CREATE 权限判定通过与否，*P* 表示通过，*R* 表示拒绝。

表 7　CREATE 权限控制结果

Table 7　CREATE permission control result

	全局	命名空间
无 CREATE 权限	*R*	*R*
有 CREATE 权限	*P*	*P*

采用以上方法可对 DML、DDL、Security、Snapshot、Namespace、Tools 和 Quotas 权限类别进行测试，测试步骤根据权限的授权级别做相应调整，例如：对于 CREATE 权限，只需测试 a)~c)和 g)步骤。

3.3.2 测试 Ⅱ

Endpoint 权限类别只包含原有的 EXECUTE 权限。本节测试只针对 EXECUTE 权限进行，该权限用于控制用户是否可以执行终端类型[11]的协处理器。

操作定义：*op2* 表示用户 *u2* 执行终端类型的协处理器，用于计算命名空间 *ns1* 中表 *t1* 的记录行数。

终端类型的协处理器需要与 HBase 的域服务器（RegionServer）直接通信，本节测试编写了 RPC 接口，以及基于 RPC 接口的客户端和服务端程序。

测试方法描述如下：

a)不授予 *u2* 任何权限，*u2* 不拥有任何角色。执行 *op2* 操作，系统应抛出类似如图 5 所示的异常信息，否则访问控制失败。

b)授予 *u2* 在全局级别上有 EXECUTE 权限，执行 *op2* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

c)撤销 *u2* 在全局级别的 EXECUTE 权限，授予 *u2* 在命名空间 *ns1* 级别有 EXECUTE 权限。执行 *op2* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

d)撤销 *u2* 在命名空间 *ns1* 级别的 EXECUTE 权限，授予 *u2* 在命名空间 *ns1* 中表 *t1* 级别有 EXECUTE 权限。执行 *op2* 操作，不抛出权限异常信息则表示访问控制成功，否则访问控制失败。

e)撤销 *u2* 所有权限，分配 *u2* 拥有角色 *role2*。用户授权替换为角色授权，重复 a)~d)测试过程。

表 8 所示为执行 *op2* 操作时，EXECUTE 权限判定通过与否，*P* 表示通过，*R* 表示拒绝。

表 8　EXECUTE 权限控制结果

Table 8　EXECUTE permission control result

	全局	命名空间	表
无 EXECUTE 权限	*R*	*R*	*R*
有 EXECUTE 权限	*P*	*P*	*P*

3.3.3 测试结果与优势分析

本文按照上述测试Ⅰ和测试Ⅱ的测试过程完成全部权限的测试工作，测试结果表明，本文设计改进的细粒度访问控制方法解决了 HBase 原有粗粒度访问控制方法的弊端，细粒度

化后的权限能真实有效地达到对用户操作请求进行访问控制的目的，提升了集群和数据库的安全性。

文献[10]的测试结果暴露出权限之间存在包含的关系，如拥有 SCAN 权限不仅可以执行 scan 操作也可以执行 get 操作。与文献[10]相比，本文设计的测试方法从权限控制级别和用户角色两个方面进行了细粒度权限测试，测试内容更全面，证明了本文所划分的细粒度权限之间不存在依赖和包含的关系，全方位验证了本文设计的细粒度访问控制方法的优越性。

3.4 时间性能分析

本文实现的细粒度访问控制方法是基于 HBase 已有的协处理器技术，通过重定制访问控制器达到控制的目的，没有额外增加控制操作过程，因此不会对 HBase 造成额外的时间性能开销。

4 结束语

本文在深入研究 HBase 源码的基础上，对 HBase 的访问控制方法进行了详细的分析，针对存在的粗粒度权限问题，给出细粒度权限划分的依据，解决了文献[9,10]没有给出划分依据或划分依据不严谨的问题。分析了 HBase 中的 RBAC 模型，通过内建数据库角色解决了细粒度权限管理难度增大的问题，能更集中管理扩展后的细粒度权限。通过扩展权限列表、重定制访问控制器，能真实有效地达到细粒度权限授权和控制的目的，有效降低了由于 HBase 原有访问控制方法导致的权限过大而可能造成的数据安全风险。本文提出的细粒度访问控制方法不会对 HBase 造成额外的性能影响。

关于 NoSQL 数据库访问控制方法的研究，目前国内外很多学者聚焦基于属性的访问控制（ABAC）方法，凸显出了 ABAC 方法在大数据应用场景下灵活多变的优势。关于如何在 HBase 集成使用 ABAC 方法是下一步研究的重点。此外，关于 HBase 行级的访问控制方法研究也极具应用价值。

参考文献：

[1] DB-Engines. DB-engines ranking [EB/OL]. [2018-07-15]. https://db-engines.com/en/ranking.

[2] Paz J R G. Microsoft Azure Cosmos DB revealed [M]. Apress, 2018.

[3] Microsoft. Azure Cosmos DB 数据库安全性 [EB/OL]. (2017-11-15) [2018-07-15]. https://docs. microsoft.com/zh-cn/azure/cosmos-db/database-security.

[4] Apache HBase Team. Apache HBase reference guide, version 1. 2. 6 [EB/OL]. (2017) [2018-07-15]. http://hbase. apache. org/book. html.

[5] Colombo P, Ferrari E. Towards a unifying attribute based access control approach for NoSQL datastores [C]//Proc of the 33rd IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE Press, 2017: 709-720..

[6] Ong K W, Papakonstantinou Y, Vernoux R. The SQL+Unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases [J]. Computer Science, 2014.

[7] Longstaff J, Noble J. Attribute based access control for big data applications by query modification[C]//Proc of the 2nd IEEE International Conference on Big Data Computing Service and Applications.Washington DC:IEEE Computer Society, 2016: 58-65.

[8] Colombo P, Ferrari E. Towards virtual private NoSQL datastores [C]// Proc of IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE Press, 2016: 193-204.

[9] 蔡平. 基于 Hadoop 的 NoSQL 数据库安全研究 [D]. 上海:上海交通大学, 2013. (Cai Ping. Research on security of NoSQL database on Hadoop [D]. Shanghai: Shanghai Jiao Tong University, 2013.)

[10] Lai Yanyan, Qian Quan. HBase fine grained access control with extended permissions and inheritable roles [C]//Proc of IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/distributed Computing. Washington DC:IEEE Computer Society, 2015: 1-5.

[11] Lai Mingjie, Koontz E, Purtell A. Coprocessor introduction [EB/OL]. (2012-02-01) [2018-07-15]. https://blogs.apache.org/hbase/entry/coprocessor_introduction.

[12] Sandhu R S, Coyne E J, Feinstein H L, *et al*. Role-based access control models [J]. Computer, 1996, 29(2): 38-47.

[13] 何海云，张春，赵战生. 基于角色的访问控制模型分析 [J]. 计算机工程, 1999,25(8): 39-41. (He Haiyun, Zhang Chun, Zhao Zhansheng. Analysis of role-based access control model [J]. Computer Engineering, 1999,25(8): 39-41.)

[14] Colombo P, Ferrari E. Enhancing MongoDB with purpose based access control [J]. IEEE Trans on Dependable & Secure Computing, 2017, PP (99): 1.

[15] Shermin M. An access control model for NoSQL databases [D]. London, Ontario, Canada: University of Western Ontario, 2013.